International Journal of Security and Networks.
Volume 2 – Numbers 3/4 – Pages 284–296.

Strong Password-Based Authentication in TLS using the Three-Party Group Diffie-Hellman Protocol

Michel Abdalla[1], Emmanuel Bresson[2], Olivier Chevassut[3], Bodo Möller[4], and David Pointcheval[1]

[1] Département d'Informatique, École normale supérieure, Paris, France
[2] Cryptology department, CELAR Technology Center, Bruz Cedex, France
[3] Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[4] Horst Görtz Institute for IT Security, Lehrstuhl für Kommunikationssicherheit, Ruhr-Universität Bochum, Bochum, Germany

Abstract. The Internet has evolved into a very hostile ecosystem where "phishing" attacks are common practice. This paper shows that the three-party group Diffie-Hellman key exchange can help protect against these attacks. We have developed password-based ciphersuites for the Transport Layer Security (TLS) protocol that are not only *provably secure* but also believed to be *free from patent and licensing restrictions* based on an analysis of relevant patents in the area.

Keywords: Password Authentication, Group Diffie-Hellman Key Exchange, TLS.

1 Introduction

An increasing number of distributed systems on the Internet are using open source software. For example, widely fielded open source software products within their respective categories are the Linux operating system, the Apache web server, the Mozilla Firefox web browser, the OpenOffice.org office suite, the OpenSSL toolkit for secure communication over the Internet, and finally the Globus toolkit[1] for building grid systems and applications over the Internet. Open source software is based on the recognition that *when programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing.*[2] This process not only produces better software but also frees users from paying royalties for running the software, for modifying and maintaining it to their liking, and even re-distributing it. At the same time the Internet has grown into an extremely hostile ecosystem.

For various open-source based Internet applications, it is highly desirable to have a cryptographic technology that allows users to securely identify themselves using short (memorable) password. The obvious approach is to use the OpenSSL implementation of the TLS protocol's unilateral-authentication mode [27] and add a straightforward password check: the server sends its X.509 certificate for explicit verification, then the user sends his password through the TLS secure channel. This is what happens in most of today's "secure" WWW applications [25]. This approach, however, provides a false sense of security because the privacy of the user's password is lost if the server is actually not authentic ("phishing" attacks), and, hence, anything protected through the password could get compromised. The password can be exposed when a user accepts a bogus server certificate, which usually takes just a single click of the mouse.

This is not all—security is in fact totally dependent on the X.509 public-key infrastructure used for server authentication: any party with a certificate apparently issued to the legitimate server can potentially obtain the password from the client. A single trusted Certification Authority (CA) that is malicious or careless could do lot of damage! Sending a one-time password in place of the fixed

[1] The Globus Alliance, `http://www.globus.org/`
[2] Open Source Initiative, `http://www.opensource.org/`

password would not help either since the bogus server could then use this password to impersonate the user.

The secure way for the user to identify himself is to tie his authentication to the TLS secure channel using some variant of the strong Password-based Authenticated Key Exchange (PAKE) primitive [1, 4, 17, 21, 24, 26, 29–31, 33–35, 37, 39, 43–45]. This primitive come in various flavors that have patent claims on them [9, 14, 32, 46] and/or are not supported by formal security arguments [35, 43–45]. The very first PAKE patent is due to Bellovin and Merritt, and is currently owned by Lucent Technologies. The Bellovin and Merritte patent entitled "Cryptographic Protocol for Secure Communications" was filed with the U.S. Patent Office in 1991 and granted in 1993 [9], and with the European Patent Office in 1992 and granted in 2002 [14]. It was also filed in various other countries such as Australia, Japan, and Canada [10, 12, 13]. This patent is a major roadblock to the adoption of a PAKE primitive by open source software.

A free software implementation of a patented PAKE primitive does not necessarily help either. The Secure Remote Protocol (SRP) from Stanford University [45, 46], used either as a standalone library[3] or directly implemented in open source products[4], may be considered problematic in that agreeing to the terms and conditions of the License Agreement Form's Indemnity clause is mandated[5]:

> *5. Indemnity*
> 5.1 LICENSEE agrees to indemnify, hold harmless, and defend STANFORD, UCSF-Stanford Health Care and Stanford Hospitals and Clinics and their respective trustees, officers, employees, students, and agents against any and all claims for death, illness, personal injury, property damage, and improper business practices arising out of the use or other disposition of Inventions(s), Licensed Patent(s), or Software by LICENSEE.

A patent is an obstacle to the implementation of a PAKE primitive in open source software *unless a patent grant permits it*. When SUN Microsystems Laboratories, for example, contributed elliptic-curve technology to OpenSSL, SUN also added a "patent peace" license provision to clarify its license grant [40] (patent issues are not explicitly addressed by the OpenSSL license). However, we do not have to passively wait many years for PAKE patents [9, 14, 32, 46] to expire[6] when "phishing" attacks over the Internet are developing at a furious rate. In the present paper we argue that it is indeed possible to develop PAKE-ciphersuites in TLS that achieve strong security notions and were written in attempt to be free from patent and licensing restrictions.

Examining the first claim of Bellovin and Merritt European Patent entitled "A Cryptographic Protocol for Secure Communications" [14] reveals that this patent describes the steps for generating a cryptographic key between *two parties A and B*:

> A method for communicating among plural parties at least a party A and a party B, each having access to an authentication signal [...]:
> a) A forms signal X based on a first signal R_A and transmits signal X to B;
> b) B receives signal X from A, and in response forms response signal Q based on a second signal R_B, and transmits signal Q to A;
> c) A receives signal Q from B;
> d) B generates the cryptographic key based on R_B and X; and
> e) A generates the cryptographic key based on R_A and Q

Claims 2 through 7 make this more specific, for example by using Diffie-Hellman public values for certain signals.

[3] Source code for the SRP protocol, `http://srp.stanford.edu/download.html`
[4] Patches enabling SRP ciphersuites in TLS and SSH, `http://srp.stanford.edu/links.html`
[5] The Licence Agreement Form for SRP, `http://otl.stanford.edu/pdf/97006.pdf`
[6] The Bellovin and Merritt U.S. patent, for example has a duration of seventeen years from the date of issuance which was 1993, thus it would expire August 31, 2010. U.S. patent law has since changed to have a patent term of twenty years from the date of filing.

Contributions The present paper takes into account the aforementioned constraints to develop *provably secure* PAKE-ciphersuites in TLS that are believed to *not infringe* the claims of Bellovin and Merritt European patent [14]. This Bellovin and Merritt patent only involves two parties Alice and Bob. We have specified a protocol for TLS authentication that uses a 3-party (dynamic) group Diffie-Hellman key exchange [18–20] with the third party being the helper. The helper party could be implemented by using an additional CPU or special-purpose cryptographic hardware such as an accelerator card, connected to the server. (The ciphersuites could as well have been defined the other way around with a user-side helper; however, it would cost an additional exponentiation which may not be well suited to low-powered devices.)

In this paper, we present a high-level description of these ciphersuites, and a security analysis in the tradition of provable security. By analogy to Abdalla *et al.*'s *simple open key exchange* ciphersuites [1], we named ours the *3-party group Simple Open Key Exchange* (TLS-3SOKE) since they run between two players (client and server) where the TLS server consists of two parties. Using 3SOKE for TLS combines the following three advantages over previous PAKE-ciphersuites in TLS [43, 44]:

1. TLS-3SOKE achieves strong security notions in the random oracle model under the computational Diffie-Hellman assumption in the formal model of Abdalla *et al.* [3]. The use of the random oracles, however, is very limited since we only model a hash function as a random oracle during the extraction of the pre-master secret from the Diffe-Hellman result. The password is used in the key derivation process so that the forward-security of TLS-3SOKE can be proved in concurrent executions and without having to restrict the size of the dictionary.
2. TLS-3SOKE is computationally efficient since it can work in an appropriate subgroup of the multiplicative group of a prime field without requiring a "safe prime" or even over elliptic curves or in other groups. Thus, the computations for 3SOKE can use smaller exponents than those required for [43, 44]. (The protocol described in [43] does not need safe primes, but it does not solely work in the subgroup. Rather, it involves an exponentiation to map arbitrary elements of $\mathbf{Z}_p^*$ to subgroup elements: the smaller the subgroup, the larger the exponent for this exponentiation. 3SOKE, in contrast, uses *only* the subgroup.) The 3-party group ciphersuites cost only one more exponentiation on the server side than Abdalla *et al.*'s 2-party *simple open key exchange* ciphersuites [1] and the Bellovin and al.'s EKE protocol.
3. TLS-3SOKE does not require conveying the user identity in the very first TLS handshake messages (known as `ClientHello`). This feature (also present in [1, 43], but not in [44]) provides additional protocol flexibility: the user does not have to specify a user identity and password before a SOKE ciphersuite has actually been negotiated, so these values can be chosen depending on the server identity transmitted by the server.

The 3SOKE ciphersuites for TLS (TLS-3SOKE) are essentially unauthenticated (dynamic) 3-party group Diffie-Hellman ciphersuites where the client's Diffie-Hellman ephemeral public value is encrypted under the password shared with the server; the encryption primitive is a mask generation function computed as the product of the message with a constant value raised to the power of the password. Full TLS handshakes negotiating a 3SOKE ciphersuite require modifications to the usual TLS handshake message flow to achieve security: the usual `Finished` messages of the TLS protocol, which are sent under the newly negotiated cryptographic parameters (after `ChangeCipherSpec`), are replaced by `Authenticator` messages, which are similar in meaning to the `Finished` messages, but must be sent under the old cryptographic parameters (namely, before `ChangeCipherSpec`)—that is, typically (in the case of an initial handshake) unencrypted. Also, while usually the client sends its `Finished` message first, here the server has to send its `Authenticator` message first. The client can only send its `Authenticator` message after having verified the server's `Authenticator`

message (to avoid dictionary attacks since otherwise a rogue server could try a brute force attack on the client's password [43]).

Organization of the Paper The paper is organized as follows. In Section 2 we present the mechanics of the 3SOKE ciphersuite for TLS (TLS-3SOKE). In Section 3, we analyze TLS-3SOKE in the framework of provable security to show that the ciphersuite indeed achieves strong notions of security. This treatment involves specifying the formal model, defining the appropriate security notions and algorithmic assumptions, and finally exhibiting a reduction from TLS-3SOKE to the computational Diffie-Hellman problem. In Section 4, we conclude the paper by discussing the use of TLS-3SOKE in the United States.

Related Work on Provable Security The first formal model of security to analyze Password-based Authenticated Key Exchange (PAKE) primitives is due to Bellare *et al.* [6] and was over the years refined to lead to the strong model of Abdalla *et al.* [3]. In this model interactions between the client and server, and the adversary occurs only via oracle queries. These queries indeed model the capabilities of the adversary in real attacks (see literature for more details [3, 6, 22].) Schemes proven secure in the model of Abdalla *et al.* are also secure in the model of Bellare *et al.* [6]; however, the reverse is not necessarily true due to the non tightness of the security reduction.

A PAKE is a key exchange [28, 42] with one [36] or two flows [15, 16] encrypted using the password as a common symmetric key. Bellare *et al.* [6, 8], Boyko *et al.* [17], and MacKenzie [17, 38] proposed and proved secure various PAKE structures. These structures were later proved forward-secure under various computational assumptions [2, 21, 22, 34]. Instantiations for the encryption primitive were either a password-keyed symmetric cipher or a mask generation function computed as the product of the message with the hash of a password, until Abdalla *et al.* proposed the Simple PAKE (SPAKE)-structure with a new mask generation function computed as the product of the message with a constant value raised to the power of the password [1, 4]. Whereas earlier mask generation functions need a full-domain hash function into the group, SPAKE provides high flexibility in the choice of groups (e.g., this makes it easy to work with elliptic curves).

Security researchers have also tried to use their PAKE structures for TLS authentication [1, 43, 44]. These cryptographic algorithms, however, are not supported by formal security arguments, and/or have patent claims of various breadth in certain countries [9–14, 46].

2 TLS-3SOKE Ciphersuites

Figure 1 illustrates the full TLS handshake for the case of our TLS ciphersuites. Since the abbreviated handshake for resuming a previously negotiated session is performed exactly as in other TLS ciphersuites, we only discuss the details of the full handshake.

2.1 The Handshake

Choose Ciphersuite The client and the server negotiate the ciphersuite to use and exchange nonces.

a) The client sends its list of supported ciphersuites, including our TLS ciphersuites, in the TLS initial `ClientHello` message. It includes a nonce N_c. (In the TLS specification, this nonce is known as `ClientHello.random`.)
b) The server specifies the ciphersuite to be used, selected from the client's list, by using a TLS `ServerHello` message. It also includes a nonce N_s (known as `ServerHello.random`).

Note. Each 3SOKE ciphersuite specifies a symmetric encryption algorithm (AES-CBC with either 128-bit or 256-bit keys) to use once the handshake has completed, similar to other ciphersuites. The protocol specification also provides for an integrity algorithm (HMAC-SHA1 with the current versions of TLS). Also, appropriate prime-order groups are standardized in the ciphersuite specification; they are equipped with two generators g and U that have been generated verifiably pseudo-randomly to provide assurance that no-one knows $\log_g U$ (see [5]). For this verifiably pseudo-random parameter generation, a simple variant of the hash-based procedure intended for standardization for the Digital Signature Standard [41, Appendix A.1] is used: The parameters are derived from a seed bitstring, which is also provided by the specification. (TLS-3SOKE will always use these standardized groups. The server cannot specify an alternative group since this would impose a noticeable additional burden on the client for verifying such parameters on the fly.)

Compute Diffie-Hellman Secret

a) First, the helper generates the random private exponent x_1 and computes the public key $Y_1 = g^{x_1}$. The helper hands the public key over to the server.
b) The server generates the random private exponent x_2 and computes the public key $Y_2 = \{g^{x_1}, g^{x_2}, g^{x_1 x_2}\}$. The public key is sent to the client in the form of a `ServerKeyExchange` message along with the server's identity.
c) If the server holds a private key and certificate suitable for signing, then in specific ciphersuites the server additionally sends this certificate to the client in a `Certificate` message. In this case, the `ServerKeyExchange` message additionally contains a signature on the ephemeral Diffie-Hellman public key made with the server's long-term key—similar to non-anonymous standard Diffie-Hellman ciphersuites in TLS (cf. [27, Section 7.4]).
d) The server then sends an empty message in the form of a `ServerHelloDone` to indicate that the "hello" phase is completed.
e) If the server has sent a `Certificate` message, the client verifies the signature.
f) The client generates its random private exponent x_3 and computes its public key $Y_3 = g^{x_1 x_3}$. This is precisely the dynamic group Diffie-Hellman key exchange protocol[7] [18, 19, 23] wherein the third player calls the deletion algorithm to remove the helper from the 3-party group. (Alternatively the client could have chosen not to remove the helper from the group but to form the group consisting of the helper, the server, and the client. In this case, and as specified in the patent application [23], the public key would be $Y_3 = \{g^{x_1 x_3}, g^{x_2 x_3}\}$.) Then the client encrypts the public key (using a password) as the product of the key with the password-based mask: $Y_3^\star = Y_3 \times U^{pw}$. The client encapsulated its name and the encrypted value in the `ClientKeyExchange` message which is sent out.
g) The two parties can now compute the common Diffie-Hellman secret $Z = g^{x_1 x_2 x_3}$ from the values received in the messages `ClientKeyExchange` and `ServerKeyExchange`.

Note. In the protocol description, C and S are strings giving the client identity and server identity, respectively (i.e., a user name and a server or "realm" name). The password for the user denoted by C is a string pw. Observe that the client identity C and the password pw are not used within the protocol before the server has chosen the ciphersuite and transmitted the server identity S. This means that a user only needs to provide C and pw after seeing that a handshake with S is going on. For example, HTTP servers might required password-based ciphersuites only for specific subtrees of their URL space by requesting a TLS renegotiation if necessary, and the server identity might depend on the specific URL.

[7] For [23], no patent application was filed outside of the United States.

A detail not shown in the figure is that the server, with its `ServerKeyExchange` message, will also send an index into the list of standardized groups for our ciphersuite (such as $0, 1, 2, \ldots$ for standardized 1024-bit, 1536-bit, 2048-bit, ... prime moduli with appropriate subgroups). The client should display the server's choice of group to the client, and the user should provide his password only if he agrees with the group.

Note that when the client receives Y_2 from the server and when the server receives $Y_3^\star$ from the client, it is implicit that the respective recipient performs a group membership test: it is a fatal TLS handshake failure if Y_2 or $Y_3^\star$ is not a group member.

Compute Pre-Master Secret and Authentication Key

a) The parties extract the randomness in the Diffie-Hellman result to form the pre-master secret as: $\mathsf{PreMasterSecret} = \mathsf{Hash}(C, S, pw, Y_2 \| Y_3^\star \| Z)$.
b) The pre-master secret is used as a means to derive the authentication key $\mathsf{AuthKey}$ used by the parties to perform the mutual authentication; we define $\mathsf{AuthKey} = \mathsf{PRF}_1(\mathsf{PreMasterSecret}, N_c \| N_s)$. This key derivation is performed based on the standard TLS pseudo-random function PRF (see [27, Section 5]). The key derivation function $\mathsf{PRF}_1(\mathsf{PreMasterSecret}, z)$ used here is specific to 3SOKE ciphersuites; its value is obtained as $\mathsf{PRF}(\mathsf{PreMasterSecret}, \textit{"authentication key"}, z)$.

Note. Here the randomness extractor function $\mathsf{Hash}(z_1, z_2, z_3, z_4)$ is defined as a function with multiple inputs. The reason for this is that while we assume that group elements can be represented as fixed-length strings, the same does not hold for the strings C, S, and pw. A function $\mathsf{Hash}_1(z)$ in a single input can be used to implement Hash by defining

$$\mathsf{Hash}(z_1, z_2, z_3, z_4) = \mathsf{Hash}_1\big(\mathsf{SHA1}(z_1) \| \mathsf{SHA1}(z_2) \| \mathsf{SHA1}(z_3) \| z_4\big).$$

The function $\mathsf{Hash}_1(z)$, in turn, can be instantiated by using

$$\mathsf{SHA1}(\text{constant}\langle 0 \rangle \| z) \parallel \mathsf{SHA1}(\text{constant}\langle 1 \rangle \| z) \parallel \ldots .$$

Other ways to instantiate Hash_1 are discussed in [7].

Compute Authenticators

a) Both parties use $\mathsf{AuthKey}$ to produce the authenticators Auth_C and Auth_S. More precisely, the authenticator is set as a MAC on "*finished_label* $\|$ *hash of handshake*", in which *finished_label* is the string "client finished" or "server finished", depending on which party is sending the respective message, and where *hash of handshake* denotes the hash of the concatenation of all the handshake messages sent so far in both directions exactly as would be used for the `Finished` message in other TLS ciphersuites (i.e., the MD5 hash concatenated with the SHA-1 hash). The server sends its `Authenticator` message first.
b) The client first checks the server's authenticator, and, if it is correct, sends its own `Authenticator`, and then proceeds to the `ChangeCipherSpec` message.
c) The server subsequently checks the client's authenticator, and, if correct, replies by sending a `ChangeCipherSpec` message as well.

Note. Usual TLS ciphersuites send `Finished` messages for authentication *after* switching to the newly negotiated key material ($\mathsf{KeyBlock}$) in the TLS record layer (which event is indicated by a `ChangeCipherSpec` message). This approach, however, would violate the security notion that 3SOKE ciphersuites are designed to achieve. Instead, 3SOKE ciphersuites make use of

`Authenticator` messages; the client does not send its `Authenticator` and `ChangeCipherSpec` before it has verified the server's `Authenticator`, and the server delays its `ChangeCipherSpec` until it has verified the client's `Authenticator`. Note that the `Authenticator` message does not undergo any processing using the KeyBlock, since it precedes the `ChangeCipherSpec`; this is different from the handshake in other TLS ciphersuites where `Finished` is sent in an TLS record processed under key material KeyBlock (see [27, Section 7.4]).

Compute Master Secret and Key Material The material KeyBlock is the bit string assigned to the Initialization Vectors (IVs), MAC secrets, and encryption keys which will protect the application sensitive messages. KeyBlock, exactly as in other TLS ciphersuites, is obtained indirectly, in two steps:

a) Both parties compute a common MasterSecret, using a function $\mathsf{PRF}_2(\mathsf{PreMasterSecret}, z)$ that is defined as $\mathsf{PRF}(\mathsf{PreMasterSecret}, \text{“master secret”}, z)$.
b) The MasterSecret is then used to obtain KeyBlock as a function $\mathsf{PRF}_3(\mathsf{MasterSecret}, z)$, which is defined as $\mathsf{PRF}(\mathsf{MasterSecret}, \text{“key expansion”}, z)$. This two-stage derivation process is used by TLS session resumption: a new connection with new client and server nonces N_c and N_s can continue to use a previously negotiated MasterSecret and derive a new KeyBlock.

Note. Another change from the standard TLS handshake message flow, besides having `Authenticator` sent before `ChangeCipherSpec`, is that for 3SOKE the server and not the client provides its authentication message first. This is necessary to protect clients against dictionary attacks: if the client was to make the start, its `Authenticator` message could be used by a malicious server that does not know pw to try out different passwords in an off-line attack, looking which one results in the `Authenticator` message as observed.

2.2 The Nitty-Gritty Details

Password derivation Remember that the protocol should allow using one out of a set of different groups (the client likely wants to see the group being used before typing his password). So for group i, of order q_i, the effective password pw may be defined in the form $pw_i = \mathsf{hash}(i \| password) \bmod q_i$.

The password string *password* actually should be hashed before being used as an exponent pw. If the protocol is changed again to use (C, S, pw) instead of just *password* to derive the exponent, we obtain a security improvement in practice in that a client can use the same *password* (string) with multiple servers and yet the respective secrets (effective passwords) will be different. (However, every server could then run a dictionary attack to recover the common underlying password.)

Exponent derivation Since the password pw appears as an exponent in the computations for TLS-3SOKE ciphersuites, some additional hash is needed to obtain this exponent from the password string *password*. In the protocol description, we do not care about details of the hash and simply use the hash result pw (in the exponent space) as the “effective password” instead: anyone knowing pw is actually able to impersonate the client or the server, and the security proof shows that attacking the protocol reduces to finding pw. In other words, at the protocol level, pw is the password needed for authentication and *password* is just a way to remember it.

Using U^{pw} as the effective password A possible variant of 3SOKE is as follows. From the description in Section 2.1, it can be noticed that, if we modify the input to Hash to use U^{pw} instead of pw, the server does not really have to store pw, it can store just U^{pw} instead. Here U^{pw} becomes

the "effective password" because the client too only needs U^{pw} in its computations (whereas the physical user, of course, will still memorize and type the short password). Unfortunately, the security proof for this new scheme cannot be reduced to the CDH problem, because knowledge of the password pw used in the query to the random oracle Hash is needed to perform the reduction. There are, however, ways to circumvent this problem. We describe the most natural ones here.

One way is to assume small (polynomial-sized) dictionaries such that pw can be extracted from the value U^{pw} using an exhaustive search algorithm. With pw in hand the proof can then be performed as before. Such solution is, however, quite constraining from the computational point of view (N exponentiations are needed). Also note that when restricting to polynomial-sized dictionaries, one must avoid U^{pw} being known by the adversary, since otherwise a trivial off-line attack can be mounted.

Another way to preserve provable security while using U^{pw} in the Hash is to use a random oracle $\mathcal{G}$ when computing pw from the password string $password$: that is, the effective password is $U^{\mathcal{G}(password)}$. In this latter case, one could make use of the calls to the random oracle $\mathcal{G}$ to obtain the value pw in order to complete the proof. Briefly speaking, for a given value θ being used as input to the random oracle Hash, we have to test for each π output by $\mathcal{G}$ whether $\theta = U^{\pi}$. Thus, the impact on the reduction would be an increase in the total computational time by an additive factor $q_{\mathcal{G}} \cdot \tau_e$, where $q_{\mathcal{G}}$ denotes the total number of calls to the the random oracle $\mathcal{G}$ and τ_e denotes the computational time for an exponentiation in $\mathbb{G}$.

3 Provable Security Results

3.1 Model and Security Notions for 2-Party

Model A password-based authenticated key-exchange protocol $\mathbf{P}$ runs between a *client* $C \in \mathsf{client}$ and a *server* $S \in \mathsf{server}$. Both the client and the server can have several *instances* involved in distinct, possibly concurrent executions of the protocol. (An instance i, often termed oracle, of a client or a server is referred to as P^i.) A client C holds a password pw_C and a server S holds the *derived password* $pw_S[C]$ for this client [6]. (Protocols wherein $pw_S[C] = pw_C$ are called symmetric; in general, $pw_S[C]$ may differ from pw_C. The value pw_C and $pw_S[C]$ are often termed the long-lived keys of the client and the server respectively.) Each password pw_C is drawn from the dictionary $\mathsf{Password}$ of size N according to the uniform distribution and, therefore, is a low-entropy string.

AKE Security In order to define the privacy (semantic security) of the session key, often termed *authenticated key exchange* (AKE) security, we consider a game wherein the protocol $\mathbf{P}$ is executed in the presence of the adversary $\mathcal{A}$. In this game $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, \mathbf{P})$, we draw a password pw from $\mathsf{Password}$, provide coin tosses to the adversary, and give the adversary access to the oracles via the following three oracle queries:

- $\mathsf{Execute}(C^i, S^j)$: The output of this query consists of the messages exchanged during the honest execution of the protocol. This models passive attacks.
- $\mathsf{Send}(P^i, m)$: The output of this query is the message that the instance P^i would generate upon receipt of message m. A query $\mathsf{Send}(P^i, \text{"start"})$ initializes the key exchange protocol, and thus the adversary receives the initial flow that the initiator would send to the receiver. This models active attacks.
- $\mathsf{Test}(P^i)$: This query tries to capture the adversary's ability to distinguish real keys from random ones. In order to answer it, we need a private random coin b (unique for the *whole* game) and then forward to the adversary either the session key sk held by P^i if $b = 1$ or a random key of the same size if $b = 0$. We emphasize that the adversary is allowed to ask several Test-queries.

The goal of the adversary in $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, \mathbf{P})$ is to guess the hidden bit b involved in the Test-queries, by outputting a guess b'. Let Succ denote the event in which the adversary is successful and correctly guesses the value of b. The **AKE advantage** of an adversary $\mathcal{A}$ is then defined as $\mathsf{Adv}^{\mathsf{ake}}_{\mathbf{P}}(\mathcal{A}) = 2\Pr[\mathsf{Succ}] - 1$. The protocol $\mathbf{P}$ is said to be (t, ε)-**AKE-secure** if $\mathcal{A}$'s advantage is smaller than ε for any adversary $\mathcal{A}$ running with time t. Note that the advantage of an adversary that simply guesses the bit b is 0 in the above definition due to the rescaling of the probabilities.

MA Security The notion of *mutual authentication* (MA) is also often of protocol. A protocol $\mathbf{P}$ is said to achieve MA if each party can be ensured that it has established a session key with the intended players. In the context of key-exchange protocols, authentication between the players is often achieved via *authenticators*. Intuitively, an authenticator is a value that can only be computed (except with small probability) with the knowledge of a secret. The idea is that if a party has sent data in the clear, it must subsequently provide an authenticator relative to these data. We denote by $\mathsf{Succ}^{\mathsf{auth}_S}(\mathcal{A})$ the success probability of an adversary trying to impersonate the server (i.e., the probability that a client will finish the key exchange protocol accepting the adversary as an authenticated server).

FS Security Another security notion to look at is *forward-secrecy* (FS). A protocol $\mathbf{P}$ is said to achieve forward-secrecy if the security of a session key between two participants is preserved even if one of the two participants later is compromised. In order to consider forward-secrecy, one has to account for a new type of query, the $\mathsf{Corrupt}$-query, which models the compromise of a participant by the adversary:

– $\mathsf{Corrupt}(P)$: This query returns to the adversary the long-lived password pw_P for participant P. As in [6], we assume the weak corruption model in which the internal states of all instances of that user are not returned to the adversary.

Let Succ denote the event in which the adversary successfully guesses the hidden bit b used by Test oracle. The **FS-AKE advantage** of an adversary $\mathcal{A}$ is then defined as $\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathbf{P}}(\mathcal{A}) = 2\Pr[\mathsf{Succ}] - 1$. (Defining $\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathbf{P}}(\mathcal{A})$ here means relaxing a restriction in $\mathbf{Game}^{\mathsf{ake}}(\mathcal{A}, \mathbf{P})$: the real session keys are used to answer to the Test-queries after a $\mathsf{Corrupt}$-query.) The protocol $\mathbf{P}$ is said to be (t, ε)-**FS-AKE-secure** if advantage $\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathbf{P}}(\mathcal{A})$ is smaller than ε for any adversary $\mathcal{A}$ running with time t.

3.2 Assumptions

$\mathsf{CDH}_{g,\mathbb{G}}$ A (t, ε)-$\mathsf{CDH}_{g,\mathbb{G}}$ attacker, in a finite cyclic group $\mathbb{G}$ of prime order q with g as a generator, is a probabilistic machine Δ running in time t such that its success probability $\mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(\Delta)$, given random elements g^x and g^y to output g^{xy}, is greater than ε:

$$\mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(\Delta) = \Pr[\Delta(g^x, g^y) = g^{xy}] \geq \varepsilon.$$

We denote by $\mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t)$ the maximal success probability over every adversaries running within time t. The CDH-Assumption states that $\mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t) \leq \varepsilon$ for any t/ε not too large.

PRFs A pseudo-random function family (PRF) is a family of functions $\mathcal{F} = (f_k)_k$ in $\mathcal{F}_{m,n}$, the set of the functions from $\{0,1\}^m$ into $\{0,1\}^n$, indexed by a key $k \in \{0,1\}^\ell$, so that for a randomly chosen ℓ-bit key k, no adversary can distinguish the function f_k from a truly random function in $\mathcal{F}_{m,n}$: we

define the advantage $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{D}, q) = |\Pr_k[1 \leftarrow \mathcal{D}^{f_k}] - \Pr_f[1 \leftarrow \mathcal{D}^{f}]|$, where $\mathcal{D}$ is a distinguisher, with an oracle access to either a random instance f_k in the given family $\mathcal{F}$ or a truly random function f in $\mathcal{F}_{m,n}$, and must distinguish the two cases with at most q queries to the function oracle. We say that such a family is a (q, ε, t)-PRF if for any distinguisher asking at most q queries to the oracle, its advantage is less than ε, after a running time bounded by t.

MAC A Message Authentication Code, say $\mathsf{MAC} = (\mathsf{MAC.Sign}, \mathsf{MAC.Verify})$, is made of the two following algorithms, with a secret key sk uniformly distributed in $\{0,1\}^{\ell_{\mathrm{M}}}$:

- The *MAC generation algorithm* MAC.Sign. Given a message m and secret key $sk \in \{0,1\}^{\ell_{\mathrm{M}}}$, MAC.Sign produces an authenticator μ. This algorithm might be probabilistic.
- The *MAC verification algorithm* MAC.Verify. Given an authenticator μ, a message m and a secret key sk, MAC.Verify tests whether μ has been produced using MAC.Sign on inputs m and sk.

The classical security level for MAC is to prevent existential forgeries, even for an adversary which has access to the generation and the verification oracles. We denote by $\mathsf{Succ}^{\mathsf{mac}}(t, q)$ the maximum success probability of a forger running within time t and making at most q queries to the MAC.Sign oracle.

3.3 Security Results

Theorem 1 (FS-AKE Security). *Let us consider protocol from Section 2.1 over a group of prime order q, where* Password *is a dictionary of size N, equipped with the uniform distribution. Let $\mathcal{A}$ be an adversary running within a time bound t that makes less than q_{active} active sessions with the parties and q_{passive} passive eavesdropping queries, and asks q_{hash} hash queries. Then we have, first $\mathsf{Succ}^{\mathsf{auth}_S}_{\mathsf{3SOKE}}(\mathcal{A})$ upper bounded by*

$$\begin{aligned}
&\frac{2q_{\mathsf{active}}}{N} + (4q_{\mathsf{session}}q_{\mathsf{hash}} + q_{\mathsf{session}} + 1)q_{\mathsf{hash}} \times \mathsf{Succ}^{\mathsf{cdh}}(t + 2\tau_e) \\
&+2\frac{{q_{\mathsf{hash}}}^2}{2^{\ell_{\mathrm{M}}}} + 2{q_{\mathsf{hash}}}^2 \times \mathsf{Succ}^{\mathsf{mac}}(t, 0) + 2q_{\mathsf{session}} \times \mathsf{Adv}^{\mathsf{prf}}(t, 2) \\
&+\frac{{q_{\mathsf{active}}}^2}{2q} + \frac{{q_{\mathsf{passive}}}^2}{2q^3} + \frac{{q_{\mathsf{hash}}}^2}{2^{\ell+1}}
\end{aligned}$$

and then $\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathsf{3SOKE}}(\mathcal{A})$ upper-bounded by

$$\begin{aligned}
&\frac{6q_{\mathsf{active}}}{N} + 2(3q_{\mathsf{session}}q_{\mathsf{hash}} + q_{\mathsf{session}} + 1)q_{\mathsf{hash}} \times \mathsf{Succ}^{\mathsf{cdh}}(t + 2\tau_e) \\
&+6\frac{{q_{\mathsf{hash}}}^2}{2^{\ell_{\mathrm{M}}}} + 6{q_{\mathsf{hash}}}^2 \times \mathsf{Succ}^{\mathsf{mac}}(t, 0) + 8q_{\mathsf{session}} \times \mathsf{Adv}^{\mathsf{prf}}(t, 2) \\
&+\frac{{q_{\mathsf{active}}}^2}{q} + \frac{{q_{\mathsf{passive}}}^2}{q^3} + \frac{{q_{\mathsf{hash}}}^2}{2^{\ell}}
\end{aligned}$$

where τ_e denotes the computational time for an exponentiation in $\mathbb{G}$.

Proof. We are interested in the event S, which occurs if the adversary correctly guesses the bit b involved in the Test-queries. We furthermore consider server (unilateral) authentication: event A is set to true if a client instance accepts, without any server partner. Let us remember that in this attack game, the adversary is allowed to use Corrupt-queries.

Game $\mathbf{G}_0$: This is the real protocol, in the random-oracle model:

$$\mathsf{Adv}^{\mathsf{ake-fs}}_{\mathsf{3SOKE}}(\mathcal{A}) = 2\Pr[\mathsf{S}_0] - 1 \quad \mathsf{Adv}^{\mathsf{auth}_S}_{\mathsf{3SOKE}}(\mathcal{A}) = \Pr[\mathsf{A}_0].$$

Let us furthermore define the event $\mathsf{S}w/t\mathsf{A} = \mathsf{S} \wedge \neg\mathsf{A}$, which means that the adversary wins the *Real-Or-Random* game without breaking authentication.

Game $\mathbf{G}_1$: In this game, we simulate the hash oracles (Hash, but also an additional hash function $\mathsf{Hash}' : (\{0,1\}^\star)^3 \to \{0,1\}^\ell$ that will appear in the Game $\mathbf{G}_3$) as usual by maintaining hash lists Λ_{Hash} and $\Lambda_{\mathsf{Hash}'}$ with all the queries-answers asked to the hash functions. We also simulate all the instances, as the real players would do, for the Send-queries and for the $\mathsf{Execute}$, Test and $\mathsf{Corrupt}$-queries.

Game $\mathbf{G}_2$: We cancel games in which some collisions appear on the transcripts $(C, S, Y_2, Y_3^\star)$, and on the master secrets. Regarding the transcripts, the distance follows from the birthday paradox since at least one element of each transcript is generated by an honest participant (at least one of them in each of the q_{active} active attacks, and all of them in the q_{passive} passive attacks). Likewise, in the case of the master keys, a similar bound applies since Hash is assumed to behave like a random oracle (which outputs ℓ-bit bit-strings):

$$\Pr[\mathsf{Coll}_2] \leq \frac{{q_{\mathsf{active}}}^2}{2q} + \frac{{q_{\mathsf{passive}}}^2}{2q^3} + \frac{{q_{\mathsf{hash}}}^2}{2^{\ell+1}}.$$

Game $\mathbf{G}_3$: In this game, we show that the success probability of the adversary is negligible in passive attacks via $\mathsf{Execute}$-queries. To do so, we modify the way in which we compute the pre-master secret $\mathsf{PreMasterSecret}$ in passive sessions that take place before or after a $\mathsf{Corrupt}$-query. More precisely, whenever the adversary asks a $\mathsf{Execute}$-query, we compute the pre-master secret $\mathsf{PreMasterSecret}$ as $\mathsf{Hash}'(C, S, Y_2 \| Y_3^\star)$ using the private oracle Hash' instead of the oracle Hash. As a result, it holds that any value of $\mathsf{PreMasterSecret}$ computed during a passive session becomes completely independent of Hash and Z, which are no longer needed in these sessions. Please note that the oracle Hash is still being used in active sessions.

The games $\mathbf{G}_3$ and $\mathbf{G}_2$ are indistinguishable unless the adversary $\mathcal{A}$ queries the hash function Hash on $(C, S, pw, Y_2 \| Y_3^\star \| Z)$, for such a passive transcript: this (bad) event is denoted $\mathsf{AskH\text{-}Passive\text{-}Exe}$. In order to upper-bound the probability of this event, we consider an auxiliary game $\mathbf{G}_3$', using a $\mathsf{CDH}_{g,\mathbb{G}}$-instance (U, V) as input, in which the simulation of the players changes—but the distributions remain perfectly identical (therefore, $\Pr[\mathsf{AskH\text{-}Passive\text{-}Exe}_3] = \Pr[\mathsf{AskH\text{-}Passive\text{-}Exe}'_3]$): Since we do not need to compute Z for the simulation of $\mathsf{Execute}$-queries, we can simulate Y_2 as $\{g^{x^\star}, V, V^{x^\star}\}$ and $Y_3^\star$ as $g^{y^\star}$, for known values of $x^\star$ and $y^\star$. This implicitly sets x_2 to be $\log_g V$, and U is still used for the mask. If event $\mathsf{AskH\text{-}Passive\text{-}Exe}$ occurs, the value $Z = \mathsf{CDH}_{g,\mathbb{G}}(g^{y^\star}/U^{pw}, V)$ can be extracted from Λ_{Hash}, by simply choosing at random among the q_{hash} elements. Since this value equals $V^{y^\star}/\mathsf{CDH}_{g,\mathbb{G}}(U, V)^{pw}$, the values $y^\star$ and pw are known, and the computation of $\mathsf{CDH}_{g,\mathbb{G}}(U, V)$ is assumed to be difficult, we get the following upper-bound:

$$\Pr[\mathsf{AskH\text{-}Passive\text{-}Exe}_3] \leq q_{\mathsf{hash}} \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + 2\tau_e).$$

Game $\mathbf{G}_4$: In this game, we consider passive attacks via Send-queries, in which the adversary simply forwards the messages it receives from the oracle instances. More precisely, we replace Hash by Hash' when computing the value of $\mathsf{PreMasterSecret}$ whenever the values $(C, S, Y_2, Y_3^\star)$ were generated by oracle instances. Note that we can safely do so due to the absence of collisions in the transcript. Like in $\mathbf{G}_3$, any value $\mathsf{PreMasterSecret}$ computed during such passive sessions becomes completely independent of Hash and Z.

As in previous games, we can upper-bound the difference in the success probabilities of $\mathcal{A}$ in games $\mathbf{G}_4$ and $\mathbf{G}_3$ by upper-bounding the probability that $\mathcal{A}$ queries the hash function Hash on $(C, S, pw, Y_2\|Y_3^\star\|Z)$, for such a passive transcript; we call this (bad) event $\mathsf{AskH\text{-}Passive\text{-}Send}$. Toward this goal, we consider an auxiliary game $\mathbf{G}_4$', in which the simulation of the players changes slightly without affecting the view of the adversary. In this simulation, we are given a $\mathsf{CDH}_{g,\mathbb{G}}$-instance (U, V), and choose at random one of the $\mathsf{Send}(S, \text{"start"})$-queries being asked to S and we reply with $Y_2 = \{g^{x^\star}, V, V^{x^\star}\}$ for a known, random $x^\star$. On the client side, we change the simulation whenever it receives as input a message that was generated by a server instance, by computing $Y_3^\star$ as $g^{y^\star}$ for a known, random $y^\star$. If the event $\mathsf{AskH\text{-}Passive\text{-}Send}$ occurs and our guess for the passive session is correct (the adversary simply forwarded the messages), then we can extract $Z = \mathsf{CDH}_{g,\mathbb{G}}(g^{y^\star}/U^{pw}, V) = V^{y^\star}/\mathsf{CDH}_{g,\mathbb{G}}(U, V)^{pw}$ from Λ_{Hash}. Similarly to above, we get:

$$\Pr[\mathsf{AskH\text{-}Passive\text{-}Send}_4] \leq q_{\mathsf{session}} q_{\mathsf{hash}} \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + 2\tau_e).$$

Game $\mathbf{G}_5$: In this game, we make one of the most significant modifications. We replace the oracle Hash by the private oracle Hash' whenever the input to this query contains an element that was not generated by an oracle instance and no $\mathsf{Corrupt}$-query has occurred. More precisely, if either the value $Y_2 = \{Y_2^{(1)}, Y_2^{(2)}, Y_2^{(3)}\}$ or $Y_3^\star$ in the Hash-query $(C, S, pw, Y_2\|Y_3^\star\|Z)$ was generated by the adversary, with $Z = \mathsf{CDH}_{g,\mathbb{G}}(Y_3^\star/U^{pw}, Y_2^{(2)})$, then we reply to this query using $\mathsf{Hash}'(C, S, Y_2\|Y_3^\star)$ as long as no $\mathsf{Corrupt}$-query has occurred. Note that we can check the value Z since we know pw and at least x_2 or x_3. Clearly, the games $\mathbf{G}_5$ and $\mathbf{G}_4$ are indistinguishable as long as $\mathcal{A}$ does not query the hash function Hash on an input $(C, S, pw, Y_2\|Y_3^\star\|Z)$, for some execution transcript $(C, S, Y_2\|Y_3^\star)$. We denote this (bad) event by $\mathsf{AskHbC\text{-}Active}$. Thus,

$$|\Pr[\mathsf{A}_5] - \Pr[\mathsf{A}_4]| \leq \Pr[\mathsf{AskHbC\text{-}Active}_5]$$
$$|\Pr[\mathsf{S}w/t\mathsf{A}_5] - \Pr[\mathsf{S}w/t\mathsf{A}_4]| \leq \Pr[\mathsf{AskHbC\text{-}Active}_5].$$

Game $\mathbf{G}_6$: In this game, we replace the pseudo-random functions by truly random functions for all the sessions in which the value of $\mathsf{PreMasterSecret}$ has been derived with the private oracle Hash'. Since the value $\mathsf{PreMasterSecret}$ that is being used as the secret key for the pseudo-random function is independently and uniformly distributed, the distance can be proven by a classical sequence of hybrid games, where the counter is on the pre-master secrets. That is, each time a new pre-master secret is set, we increment the counter. Then, $\Pr[\mathsf{S}w/t\mathsf{A}_6] = \frac{1}{2}$.

$$|\Pr[\mathsf{A}_6] - \Pr[\mathsf{A}_5]| \leq q_{\mathsf{session}} \times \mathsf{Adv}^{\mathsf{prf}}(t, 1)$$
$$|\Pr[\mathsf{S}w/t\mathsf{A}_6] - \Pr[\mathsf{S}w/t\mathsf{A}_5]| \leq q_{\mathsf{session}} \times \mathsf{Adv}^{\mathsf{prf}}(t, 2).$$

Game $\mathbf{G}_7$: In this game, we exclude collisions on MAC keys for all the sessions in which the pre-master secret $\mathsf{PreMasterSecret}$ has been derived with the private oracle Hash'(which event is denoted $\mathsf{CollPRF}$). For these sessions, the MAC keys of length ℓ_{M} are independently and uniformly distributed (because the PRF were replaced by random functions), so the probabilities differ from those in the previous game by at most:

$$\Pr[\mathsf{CollPRF}_7] \leq {q_{\mathsf{hash}}}^2/2^{\ell_{\mathrm{M}}}.$$

Game $\mathbf{G}_8$: In this game, we exclude games wherein for some transcript $(C, S, Y_2\|Y_3^\star)$, there are two passwords pw_0 and pw_1 such that the corresponding pre-master secrets lead to a collision of the MAC-values (which event is denoted CollM).

Since we know that MAC-keys are truly random and different from each other at this point, the event CollM means that a MAC with a random key (one of the q_{hash} possible values) may be

a valid forgery for another random key. Thus, by randomly choosing the two indices for the hash queries, we get the following upper-bound:

$$\Pr[\mathsf{CollM}_8] \leq {q_{\mathsf{hash}}}^2 \times \mathsf{Succ}^{\mathsf{mac}}(t, 0).$$

Before proceeding with the rest of the analysis, we split the event AskHbC-Active into two disjoint sub-cases depending on whether the adversary impersonates the client (and thus interacts with the server) or the server (and thus interacts with the client). We denote these events $\mathsf{AskHbC}_w\mathsf{S}$ and $\mathsf{AskHbC}_w\mathsf{C}$, respectively. Also we denote by $q_{\mathsf{fake-server}}$ (respectively, $q_{\mathsf{fake-client}}$) the number of sessions in which the adversary impersonates the server (resp., the client). Obviously, one has $q_{\mathsf{fake-server}} + q_{\mathsf{fake-client}} \leq q_{\mathsf{active}}$.

Game $\mathbf{G}_9$: In this game, we focus on $\mathsf{AskHbC}_w\mathsf{C}$ only. We now reject all the authenticators sent by the adversary for all the sessions in which the pre-master secret PreMasterSecret has been derived with the private oracle Hash': $\Pr[\mathsf{A}_9] = 0$. In order to evaluate the distance between the games $\mathbf{G}_9$ and $\mathbf{G}_8$, we consider the probability of the event $\mathsf{AskHbC}_w\mathsf{C}$, in which the adversary succeeds in faking the server by sending a valid authenticator to the client before a Corrupt-query.

To evaluate the probability of event $\mathsf{AskHbC}_w\mathsf{C}$, we note that, up to the moment in which a Corrupt-query occurs, no information on the password pw of a user is revealed to the adversary, despite the fact that the password is still used in the computation of $Y_3^\star$. To see that, note that, for any given transcript $(C, S, Y_2 \| Y_3^\star)$ in which $Y_3^\star$ was created by an oracle instance and for each password pw, there exists a value $x \in \mathbb{Z}_q$, such that $Y_3^\star = g^x U^{pw}$, which is never revealed to the adversary. Moreover, since we have removed collisions on the pre-master secrets, on the MAC keys, and on the MAC values, there is at most one password that can lead to a valid authenticator. As a result, the probability that the adversary succeeds in sending a valid authenticator in each of theses sessions is at most $1/N$. Thus, we get

$$\Pr[\mathsf{AskHbC}_w\mathsf{C}_9] \leq \frac{q_{\mathsf{fake-server}}}{N}.$$

Game $\mathbf{G}_{10}$: We finally concentrate on the success probability of the adversary in faking the client. What we show in this game is that the adversary cannot eliminate more than one password in the dictionary by impersonating a client. To do so, we first upper-bound the probability that, for some transcript $(C, S, Y_2 \| Y_3^\star)$ in which Y_2 was created by server instance, there are two hash queries in Λ_{Hash} such that one has $(Y_2, Y_3^\star, pw_0, Z_0 = \mathsf{CDH}_{g,\mathbb{G}}(Y_3^\star / U^{pw_0}, \hat{Y}_2))$ and $(Y_3^\star, Y_2, pw_1, Z_1 = \mathsf{CDH}_{g,\mathbb{G}}(Y_3^\star / U^{pw_1}, \hat{Y}_2))$. We denote this event CollH.

In order to upper-bound the probability of event CollH, we consider an auxiliary game in which the simulation of the players changes slightly without affecting the view of the adversary. The goal is to use the adversary to help us compute the computational Diffie-Hellman value of U and V. In this simulation, we choose at random one of the $\mathsf{Send}(S, \text{“start”})$-queries being asked to S and we reply with $Y_2 = \{g^{x^\star}, V, V^{x^\star}\}$ in the hope that this is the session which leads to a collision in the transcript. For all other sessions, Y_2 is simulated as $\{g^{x_1}, g^{x_2}, g^{x_1 x_2}\}$. Now, let us assume that the event CollH happens. If our guess for the $\mathsf{Send}(S, \text{“start”})$-query was correct, then we can extract the value $\mathsf{CDH}_{g,\mathbb{G}}(U, V)$ as $(Z_1/Z_0)^u$, where u is the inverse of $(pw_0 - pw_1)$, by simply guessing the indices of the two hash queries involved in the collision. We note that u is guaranteed to exist since $pw_0 \neq pw_1$. It follows that

$$\Pr[\mathsf{CollH}] \leq q_{\mathsf{session}} {q_{\mathsf{hash}}}^2 \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + \tau_e).$$

When the event CollH does not happen, for each transcript $(C, S, Y_2 \| Y_3^\star)$ in which Y_2 was created by server instance, there is at most one password value pw such that the tuple $(Y_2, Y_3^\star, Z =$

$\mathsf{CDH}_{g,\mathbb{G}}(Y_3^{\star}/U^{pw}, \hat{Y}_2))$ is in Λ_{Hash}. As a result, the probability of the adversary in impersonating a client reduces to trying one password at a time. Thus,

$$\Pr[\mathsf{AskHbC}_w\mathsf{S}_{10}] \leq \frac{q_{\mathsf{fake-client}}}{N} + q_{\mathsf{session}} {q_{\mathsf{hash}}}^2 \times \mathsf{Succ}^{\mathsf{cdh}}_{g,\mathbb{G}}(t + \tau_e).$$

Since $\Pr[\mathsf{A}_{10}] = 0$, this concludes the proof of Theorem 1. □

4 Discussions and Recommendations

This paper describes *efficient* and *provably secure* ciphersuites for password-based authentication in the TLS protocol. It is a first attempt at drafting *provably secure* PAKE ciphersuites for TLS that are believed to *not infringe* existing patents [9, 14, 32, 46]; however, further investigation would be needed before this technology can be used within the United States completely without fear of infringement. Also, in the same vein as SUN Microsystems Laboratories's contribution to OpenSSL, contributing the TLS-3SOKE technology to OpenSSL would require a license grant permit from the Lawrence Berkeley National Laboratory–which is not a commercial enterprise and whose main purpose is not to make a profit.

At the time of this writing, it is not totally clear that the steps defined by the claims of Bellovin and Merritt U.S. Patent involve only two parties [9]. Claim #1 through #24 appear to define the steps of the 2-party encrypted Diffie-Hellman key exchange, but use the phrase *"a plurality of parties"*. In typical U.S. patent law usage, a *"plurality"* refers to two or more. However, in the context of the patent specification, only Alice and Bob are mentioned. It should be noted that a review of the patent file history tells us that the U.S. patent required two office actions, which resulted in the amendment of all the claims. Thus, only literal infringement would be possible under current U.S. patent law. Furthermore in their corresponding European patent [14], Bellovin and Merritt were forced to explicitly call out the two parties, Alice and Bob. Arguably, the European limitations of using only Alice and Bob could be used as an estoppel argument in the U.S. case, to infer that only two parties are ever used. If this argument were successful, then three-or-more party would not infringe the U.S. Bellovin and Merritt patent.

Another argument could be made regarding the validity of the U.S. patent looking at U.S. claim #1. Here, Bellovin and Merritt transmits a signal to one of the parties, but the *"receiving a response signal"* step never states who sends the response signal. In present-day patent practice, this could be viewed as "indefinite", which could lead to an invalid and potentially unenforceable patent:

> A method for generating a cryptographic key to a first symmetric key cryptosystem by using an authentication signal, said authentication signal being available to a plurality of parties, said method compromising the steps of:
> - forming an outgoing signal by encrypting at least a portion of an excitation signal with a second symmetric key cryptosystem using a key based on said authentication signal, said excitation signal being based on first signal R_A;
> - transmitting said outgoing signal to one of said parties;
> - receiving a response signal, Q, in response to said outgoing signal; and
> - generating said cryptographic key based on said first signal and on said response signal.

Alternatively, the "receiving a response signal" element could be interpreted in the context of the Bellovin and Merritt patent specification to only mean an interchange between Alice and Bob, with no third party involved.

The arguments made above could give an impression of weakness or fear of invalidity in the patent, thus perhaps making it less likely that a patent owner would attempt to enforce the patent.

Before the open source community *at large* can benefit from the ciphersuites presented in this paper, investigations should also be performed in various other countries (e.g, Australia, Japan, Canada [10, 12, 13]) where the Bellovin and Merritt patent was also filed.

It should also be noted that the U.S. Bellovin and Merritt patent expires on August 31, 2010 after which no patent infringement issue remains in the United States.

Acknowledgments

The authors want to thank Dr. Joseph R. Milner, a Patent Attorney at Lawrence Berkeley National Laboratory, for suggesting this paper, for invaluable discussions on the patents (and licensing issues), and for providing us with the Discussions and Recommendations Section.

The authors want to thank Abdelihah Essiari for developing and implementing a prototype of this protocol within the OpenSSL open-source implementation of TLS. His work has provided us with a meaningful practical feedback.

The first and fifth authors have been supported in part by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT. The third author was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Information and Computing Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This document is report LBNL-59947. See `http://www-library.lbl.gov/disclaimer`.

References

1. Michel Abdalla, Emmanuel Bresson, Olivier Chevassut, Bodo Möller, and David Pointcheval. Provably secure password-based authentication in TLS. ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'06), pages 35–45. ACM Press, March 2006.
2. Michel Abdalla, Olivier Chevassut, and David Pointcheval. One-time verifier-based encrypted key exchange. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 47–64. Springer-Verlag, January 2005.
3. Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84. Springer-Verlag, January 2005.
4. Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 191–208. Springer-Verlag, February 2005.
5. Michel Abdalla, Emmanuel Bresson, Olivier Chevassut, Abdelilah Essiari, Bodo Möller and David Pointcheval. Simple Open Key Exchange (SOKE) ciphersuites for password authentication in TLS. Work in progress, to be published as Internet Draft. 2006.
6. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, May 2000.
7. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, November 1993.
8. Mihir Bellare and Phillip Rogaway. The AuthA protocol for password-based authenticated key exchange. Contributions to IEEE P1363, March 2000.
9. S. M. Bellovin and M. Merritt. Cryptographic protocol for secure communications. U.S. Patent #5,241,599, August 1993. Available on-line at `http://www.uspto.gov/`.
10. S. M. Bellovin and M. Merritt. A cryptographic protocol for secure communications. Australian Patent #648433B2, April 1994. Available on-line at `http://www.ipaustralia.gov.au/patents/`.
11. S. M. Bellovin and M. Merritt. Cryptographic protocol for remote authentication. U.S. Patent #5,440,635, August 1995. Available on-line at `http://www.uspto.gov/`.

12. S. M. Bellovin and M. Merritt. Protocol and apparatus for safe communication. Japanese Patent #2599871B2B2, April 1997. Available on-line at `http://www.jpo.go.jp/index.htm`.
13. S. M. Bellovin and M. Merritt. Cryptographic protocol for secure communications. Canadian Patent #2076252C, August 1998. Available on-line at `http://patents1.ic.gc.ca/intro-e.html`.
14. S. M. Bellovin and M. Merritt. A cryptographic protocol for secure communications. European Patent #0535863B1, January 2002. Available on-line at `http://my.epoline.org/portal/public`.
15. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.
16. Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 244–250. ACM Press, November 1993.
17. Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 156–171. Springer-Verlag, May 2000.
18. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 290–309. Springer-Verlag, December 2001.
19. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 321–336. Springer-Verlag, April 2002.
20. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Group Diffie-Hellman key exchange secure against dictionary attacks. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 497–514. Springer-Verlag, December 2002.
21. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In *ACM CCS 03: 10th Conference on Computer and Communications Security*, pages 241–250. ACM Press, October 2003.
22. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. New security results on encrypted key exchange. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC 2004: 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 145–158. Springer-Verlag, March 2004.
23. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Cryptography for secure dynamic group communication. U.S. Patent Application 20050157874, November 30, 2004. Available on-line at `http://www.lbl.gov/Tech-Transfer/techs/lbnl1973.html`.
24. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer-Verlag, May 2005.
25. Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Password interception in a SSL/TLS channel. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599. Springer-Verlag, August 2003.
26. Dario Catalano, David Pointcheval, and Thomas Pornin. IPAKE: Isomorphisms for password-based authenticated key exchange. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 477–493. Springer-Verlag, August 2004.
27. Tim Dierks and Christopher Allen. *RFC 2246 - The TLS Protocol Version 1.0*. Internet Activities Board, January 1999.
28. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1978.
29. Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 524–543. Springer-Verlag, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`.
30. C. Gentry, P. MacKenzie, , and Z. Ramzan. Opake: Password authenticated key exchange based on the hidden smooth subgroup assumption. ACM Computer and Communications Security, November 2005.
31. Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 408–432. Springer-Verlag, August 2001. `http://eprint.iacr.org/2000/057`.
32. D. Jablon. Cryptographic methods for remote authentication. U.S. Patent #6,226,383, January 2002. Available on-line at `http://www.uspto.gov/`.
33. David P. Jablon. Password authentication using multiple servers. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 344–360. Springer-Verlag, April 2001.

34. Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Forward secrecy in password-only key exchange protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 29–44. Springer-Verlag, September 2002.
35. T. Kwon. Authentication and key agreement via memorable password. Network and Distributed System Security (NDSS) Symposium, February 2001.
36. Stefan Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Workshop on Security Protocols*, École Normale Supérieure, 1997.
37. Philip D. MacKenzie. More efficient password-authenticated key exchange. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 361–377. Springer-Verlag, April 2001.
38. Philip D. MacKenzie. The PAK suite: Protocols for password-authenticated key exchange. Technical Report 2002-46, DIMACS, 2002.
39. Philip D. MacKenzie, Sarvar Patel, and Ram Swaminathan. Password-authenticated key exchange based on RSA. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 599–613. Springer-Verlag, December 2000.
40. Sun Microsystems. Sun Microsystems contributed elliptic curve cryptographic algorithms to open source projects, September 2002. See `http://research.sun.com/projects/crypto/`. See also `http://www.sun.com/cddl/`.
41. National Institute of Standards and Technology (NIST). Digital Signature Standard. Draft FIPS 186-3, March 2006.
42. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
43. Michael Steiner, Peter Buhler, Thomas Eirich, and Michael Waidner. Secure password-based cipher suite for TLS. *ACM Transactions on Information and System Security*, 4(2):134–157, 2001.
44. David Taylor, Tom Wu, Nikos Mavroyanopoulos, and Trevor Perrin. Using SRP for TLS authentication. IETF Internet Draft, TLS Working Group, August 19, 2004.
45. T. Wu. The secure remote password protocol. Network and Distributed System Security (NDSS) Symposium, March 1998.
46. T. J. Wu. System and method for securely logging onto a remotely located computer. U.S. Patent #6,539,479, March 2003. Available on-line at `http://www.uspto.gov/`, see also `http://stanfordtech.stanford.edu/4DCGI/docket?docket=97-006`.

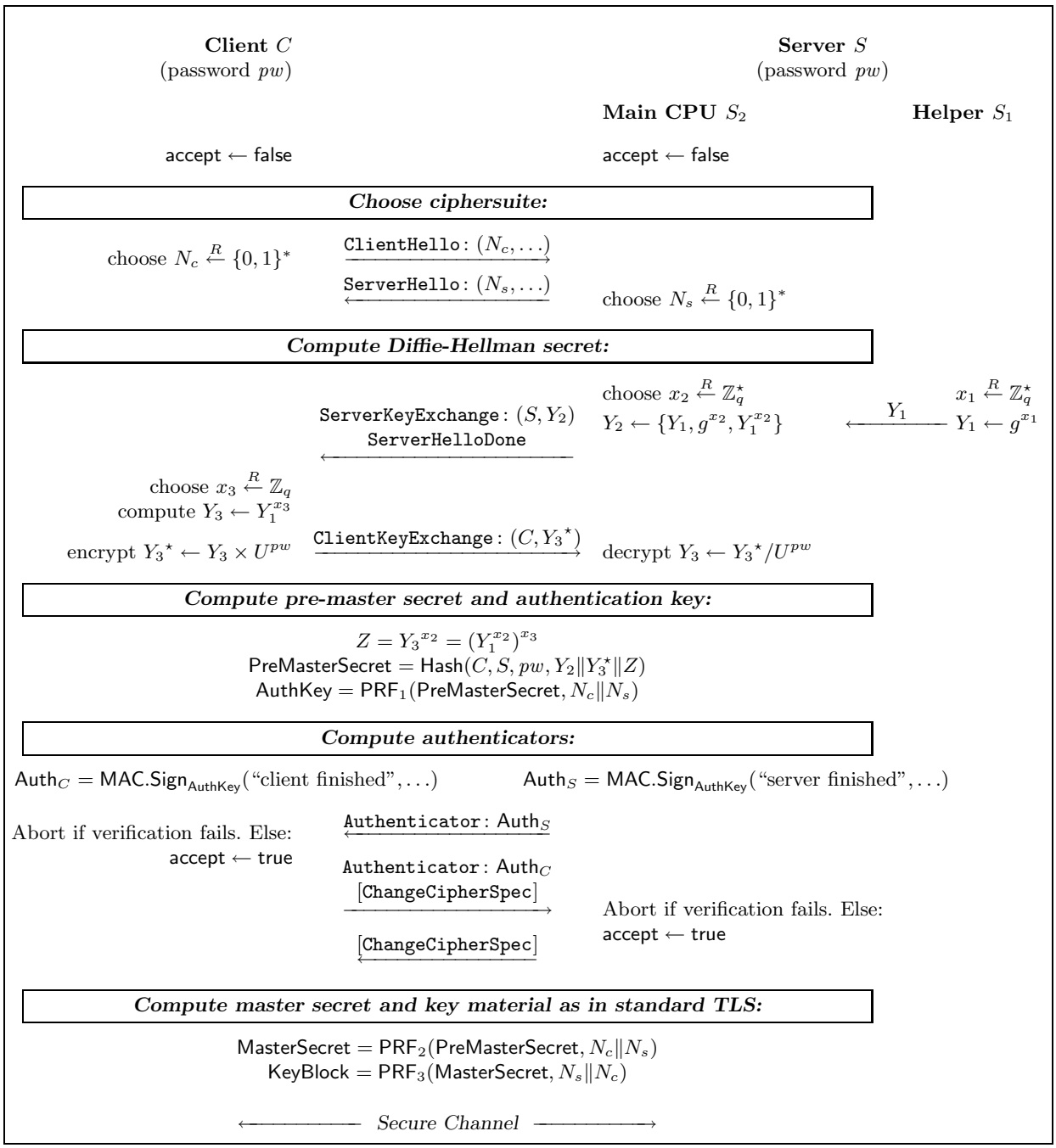

Fig. 1. The full handshake for TLS-3SOKE ciphersuites.